Exercice 1 – Exclusion mutuelle

Dans un système informatique, on dispose de trois fichiers F1, F2 et F3 et de trois processus dont les programmes A, B et C ont les structures suivantes :

Programme A	Programme B	Programme C
Actions A1	Actions B1	Actions C1 (lire F3)
Actions A2 (lire F2)	Actions B2 (écrire F3)	Actions C2
Actions A3	Actions B3 (lire F1)	Actions C3
Actions A4 (écrire F3)	Actions B4	Actions C4 (écrire F2)
Actions A5		Actions C5

Chaque fichier ne peut être lu et modifié en même temps.

1) Donner pour chaque fichier, les sections critiques de A, B et C.

2) En déduire les sections en exclusion mutuelle.

Exercice 2 - Exclusion mutuelle par variables partagées

On dispose de deux processus (P0 et P1) dont le squelette de programme est celui du cours. L'objectif de l'exercice est d'établir un algorithme utilisant des variables partagées par les processus pour réaliser l'exclusion mutuelle des sections critiques en respectant les spécifications données en cours. On ne fait aucune hypothèse sur l'atomicité des manipulations des variables.

1) Rappeler pourquoi une simple attente active sur une variable tour qui vaudrait vrai ou faux selon qu'un processus est ou non en section critique, immédiatement suivie d'une mise à jour de tour ne convient pas.

Correction :
Un processus peut être suspendu après avoir chargé tour en registre ; le deuxième chargera alors aussi une valeur d'autorisation, les deux entreront alors en sc.
(Rappel : Les spécifications de l'exclusion mutuelle sont
1. exclusion mutuelle
2. progression
3. non blocage
4. non famine (ou attente bornée)

2) On propose dans un premier algorithme, la solution suivante. La variable tour prend les valeurs 0 ou 1. L'algorithme (1) de Pi est le suivant :

```
tantque vrai faire
        actions avant sc
        tantque tour <> i faire rien
        section critique i
        tour <- 1-i
        actions apres sc
fintantque
```

(noter que 1-i = 0 si i = 1 et 1 si i = 0).

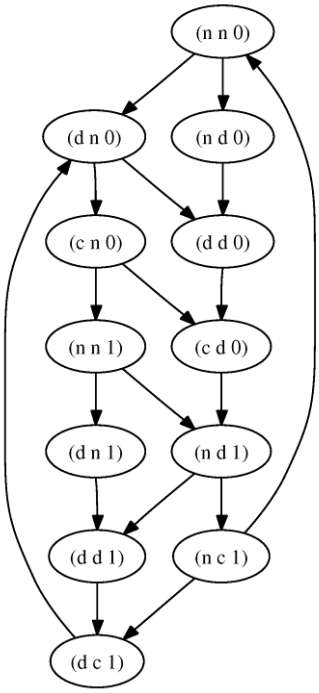
La phase d'initialisation du système est réduite à : tour ← 0. Construire le système de transitions étiquetées de cette solution. Montrer que l'exclusion mutuelle est bien réalisée mais que la condition de progression n'est pas satisfaite.

Correction :

Pour le LTS, on peut coder les états par (q0; q1; t) où qi = 1 si Pi est en sc, 0 sinon, et t la valeur de tour. Ceci constitue déjà une abstraction. Si l'on veut être exhaustif, il faut coder la position dans le code C (6 valeurs) + la valeur de tour + le compte du nombre de corps de boucle exécutés (entier tendant vers +1).

Si P0 a exécuté sa sc, il ne peut plus le faire tant que P1 n'a pas exécuté la sienne, donc une stricte alternance des sc est obligatoire => non progression.

```
// 0 : tant que tour <> i faire rien
// 1 : sc
// 2 : tour <- 1-i
```



3) Pour remédier à cet inconvénient, on emploie au lieu de tour, deux variables booléennes D0 et D1 : Di est vraie *ssi* Pi demande à passer en section critique. L'algorithme (2) de Pi est alors :

```
tantque vrai faire
        actions avant sc
        Di <- vrai
        tantque D(1-i) faire rien
```

section critique i
Di <- faux
actions apres sc
fintantque

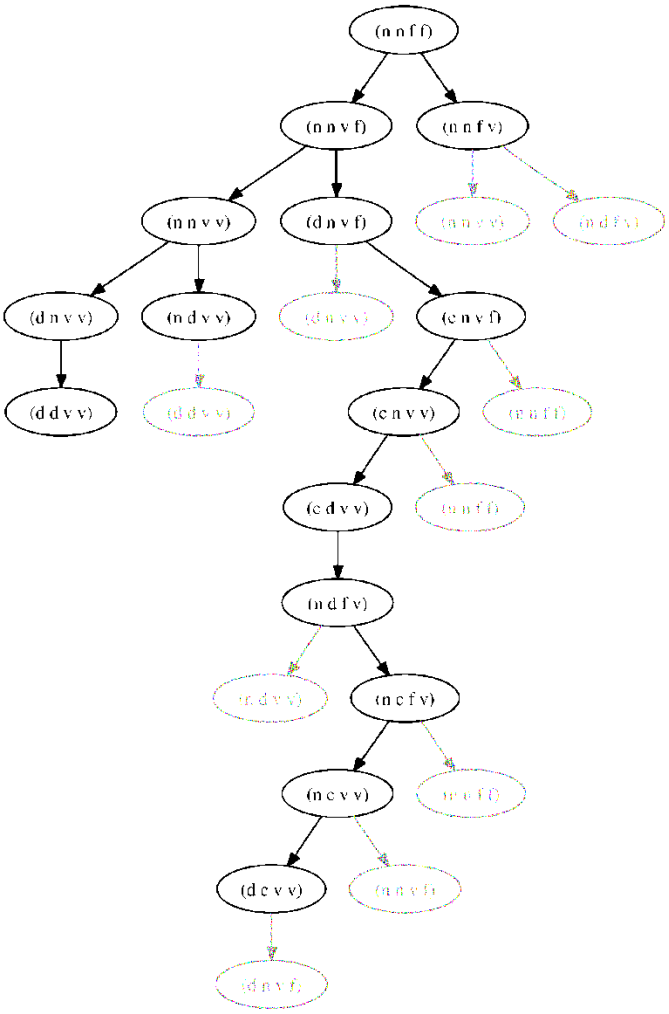
(D(1-i) désigne D1 si i = 0 et D0 si i = 1). Les variables Di sont initialisées à faux.

Construire le système de transitions étiquetées de cette solution. Montrer que la progression est maintenant assurée. Montrer que le système possède un état d'interblocage.

Correction :

Pour le LTS, on peut coder les états par (q0; q1; d0; d1) où qi = 1 si Pi est en sc, 0 sinon, et di la valeur de Di.
Il est possible que l'on obtienne D0 et D1 à vrai ! ! !

// 0 : Di <- vrai
// 1 : tant que D(i-1) faire rien
// 2 : sc
// 3 : Di <- faux



4) La dernière solution est due à G.I. Peterson (1981). Elle vérifie les propriétés attendues d'une solution d'exclusion mutuelle. On utilise une variable tour et les deux variables Di.

```
tantque vrai faire --- Algorithme de Peterson ---
        actions avant sc
        Di <- vrai
        tour <- 1-i
        tantque ( D(1-i) et (tour = 1-i) ) faire rien
        section critique i
        Di <- faux
        actions apres sc
fintantque
```

L'initialisation met tour à 0 et les Di à faux. Construire encore le système de transitions étiquetées du système. Montrer que :

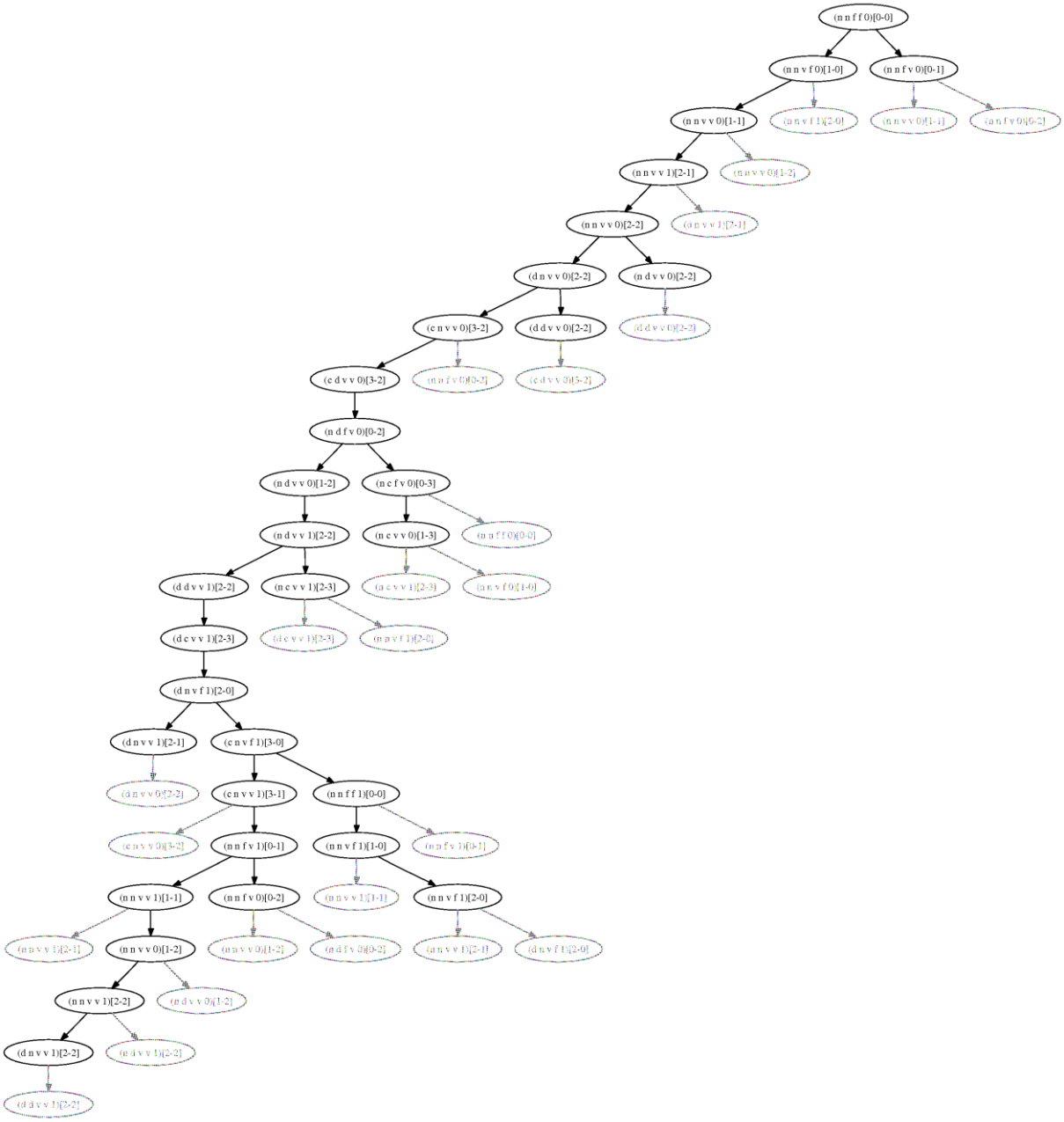
1. l'exclusion mutuelle est garantie ;
2. il n'y a pas d'interblocage ;
3. l'attente bornée est vérifiée : combien de passages en section critique de P(1−i), Pi doit-il attendre au plus lorsqu'il a demandé à passer en section critique ?

Correction :

Pour le LTS, on peut coder les états par (q0; q1; d0; d1; t) où qi = 1 si Pi est en sc, 0 sinon, di la valeur de Di et t la valeur de tour. Remarque : un LTS réduit est obtenu en codant les états de chaque processus par : n = neutre, d = demande, c = en section critique. On a ainsi au plus 32 états du système.

On attend au plus une fois. Le graphe a une jolie structure symétrique. Très joli algorithme : il a fallu attendre 1981 pour cette solution alors que le problème est posé depuis 1966 (Dijkstra).

```
// 0 : Di <- vrai
// 1 : tour <- 1-i
// 2 : tant que ( D(i-1) et (tour = 1-i) ) faire rien
// 3 : sc
// 4 : Di <- faux
```



Exercice 3 - Sémaphores

Rappeler la signification des opérations élémentaires P et V sur les sémaphores. On considère un système sur lequel s'exécutent trois processus P1, P2 et P3. P1 exécute successivement les parties de code A et B, et P2 les parties U et V. P3 ne doit commencer à réellement exécuter son code que lorsque P1 a terminé la partie A de son code. B et U sont en exclusion mutuelle.

1) Construire le graphe de précédence de ce système de processus.

P: décrémente le compteur C.S de S. Puis, si C.S < 0, le processus est bloqué.

V: incrémente le compteur de S. Puis, si C.S <= 0, un des processus bloqué sur S est réveillé.

2) Résoudre le problème entre P3 et P1 avec un sémaphore.

Utilisation d'un sémaphore pour assurer la séquentialité: P1 exécutera V(S) à la fin de A. P3 exécutera V(S) avant le début de son code. Attention à l'initialisation de C.S : à 0!!!

3) Même chose pour l'exclusion mutuelle entre P1 et P2. Peut-on utiliser le même sémaphore ?

Encadrer B et U par P(S')-V(S').
Attention à l'initialisation de C.S': à 1!!!
Et non : il faut deux sémaphores. De manière générale : ne pas chercher à « économiser » un sémaphore: on risque de perturber une solution exacte!